

[TeamCity 10.x Documentation](#) / [TeamCity Documentation](#) / [Extending TeamCity](#)

Build Script Interaction with TeamCity

Created by Christopher Hamilton, last modified by Yegor Yarko on Oct 14, 2016

If TeamCity doesn't support your testing framework or build runner out of the box, you can still avail yourself of many TeamCity benefits by customizing your build scripts to interact with the TeamCity server. This makes a wide range of features available to any team regardless of their testing frameworks and runners. Some of these features include displaying real-time test results and customized statistics, changing the build status, and publishing artifacts before the build is finished.

The build script interaction can be implemented by means of:

- service messages in the build script
- `teamcity-info.xml` file (obsolete approach, consider using service messages instead)

If you use MSBuild build runner, you can use [MSBuild Service Tasks](#).

In this section:

- [Service Messages](#)
 - [Service messages formats](#)
 - [Escaped values](#)
 - [Common Properties](#)
 - [Message Creation Timestamp](#)
 - [Message FlowId](#)
 - [Reporting Messages For Build Log](#)
 - [Blocks of Service Messages](#)
 - [Reporting Compilation Messages](#)
 - [Reporting Tests](#)
 - [Supported test service messages](#)
 - [Nested test reporting](#)
 - [Interpreting test names](#)
 - [Reporting .NET Code Coverage Results](#)
 - [Publishing Artifacts while the Build is Still in Progress](#)
 - [Reporting Build Progress](#)
 - [Reporting Build Problems](#)
 - [Reporting Build Status](#)
 - [Reporting Build Number](#)
 - [Adding or Changing a Build Parameter](#)
 - [Reporting Build Statistics](#)
 - [Disabling Service Messages Processing](#)
 - [Importing XML Reports](#)
 - [Libraries reporting results via TeamCity Service Messages](#)
- [teamcity-info.xml](#)
 - [Modifying the Build Status](#)
 - [Reporting Custom Statistics](#)
 - [Providing data using the teamcity-info.xml file](#)
 - [Describing custom charts](#)

Service Messages

Service messages are used to pass commands/build information to a TeamCity server from the build script. In order to be processed by TeamCity, they should be printed into a standard output stream of the build (otherwise, if the output is not in the service message syntax, it should appear in the build log). A single service message cannot contain a newline character inside it, it cannot span across multiple lines.

Service messages formats

Service messages support two formats:

- Single-attribute message:

```
##teamcity[<messageName> 'value']
```

- Multiple-attribute message:

```
##teamcity[<messageName> name1='value1' name2='value2']
```

Multiple attributes message can more formally be described as:

```
##teamcity[messageNameWSPpropertyNameOWSP=OWSP'value'WSPpropertyName_IDOWSP=OWSP'value'...OWSPpropertyName_IDOWSP=OWSP'value']
```

where:

- **messageName** is a name of the message. See below for supported messages. The message name should be a valid Java id (only alpha-numeric characters and "-", starting with an alpha character)
- **propertyName** is a name of the message attribute. Should be a valid Java id.
- **value** is a value of the attribute. Should be an *escaped* value (see below).
- **WSP** is a required whitespace(s): space or tab character (\t)
- **OWSP** is an optional whitespace(s)
- ... is any number of **WSPpropertyNameOWSP=OWSP'value'** blocks

Escaped values

For escaped values, TeamCity uses a vertical bar "|" as an escape character. In order to have certain characters properly interpreted by the TeamCity server, they must be preceded by a vertical bar. For example, the following message:

```
##teamcity[testStarted name='foo|'s test']
```

will be displayed in TeamCity as 'foo's test'. Please, refer to the table of the escaped values below.

| Character | Escape as |
|--|-----------|
| ' (apostrophe) | ' |
| \n (line feed) | n |
| \r (carriage return) | r |
| \uNNNN (unicode symbol with code 0xNNNN) | 0xNNNN |
| (vertical bar) | |
| [(opening bracket) | [|
|] (closing bracket) |] |

Common Properties

Any "message and multiple attribute" message supports the following list of optional attributes: timestamp, flowId. In the following examples <messageName> is the name of the specific service message.

Message Creation Timestamp

```
##teamcity[<messageName> timestamp='timestamp' ...]
```

The timestamp format is yyyy-MM-dd'T'HH:mm:ss.SSSZ or yyyy-MM-dd'T'HH:mm:ss.SSS according to [Java SimpleDateFormat syntax](#), e.g.

```
##teamcity[<messageName> timestamp='2008-09-03T14:02:34.487+0400' ...]  
##teamcity[<messageName> timestamp='2008-09-03T14:02:34.487' ...]
```

For .NET [DateTimeOffset](#), a code like this

```
var date = DateTimeOffset.Now;  
var timestamp = $"{date:yyyy-MM-dd'T'HH:mm:ss.fff}{date.Offset.Ticks:+;-}{date.Offset:hhmm}";
```

will result in

```
##teamcity[<messageName> timestamp='2008-09-03T14:02:34.487+0400' ...]
```

Message FlowId

The flowId is a unique identifier of the messages flow in a build. Flow tracking is necessary, for example, to distinguish separate processes running in parallel. The identifier is a string that should be unique in the scope of individual build.

```
##teamcity[<messageName> flowId='flowId' ...]
```

Reporting Messages For Build Log

You can report messages for a build log in the following way:

```
##teamcity[message text='<message text>' errorDetails='<error details>' status='<status value>']
```

where:

- The status attribute may take following values: NORMAL, WARNING, FAILURE, ERROR. The default value is NORMAL.
- The errorDetails attribute is used only if status is ERROR, in other cases it is ignored.

This message fails the build in case its status is ERROR and "Fail build if an error message is logged by build runner" box is checked on the [Build Failure Conditions](#) page of a build configuration. For example:

```
##teamcity[message text='Exception text' errorDetails='stack trace' status='ERROR']
```

Blocks of Service Messages

Blocks are used to group several messages in the build log.

Block opening:

The blockOpened system message allows the name attribute, you can also add a description to the blockOpened message:

```
##teamcity[blockOpened name='<blockName>' description='<this is the description of blockName>']
```

Block closing:

```
##teamcity[blockClosed name='<blockName>']
```

Please note that when you close the block, all inner blocks are closed automatically.

Reporting Compilation Messages

```
##teamcity[compilationStarted compiler='<compiler name>']  
...  
##teamcity[message text='compiler output']  
##teamcity[message text='compiler output']  
##teamcity[message text='compiler error' status='ERROR']  
...  
##teamcity[compilationFinished compiler='<compiler name>']
```

where:

- `compiler name` is an arbitrary name of the compiler performing compilation, eg. `javac`, `groovyc` and so on. Currently it is used as a block name in the build log.
- any message with status `ERROR` reported between `compilationStarted` and `compilationFinished` will be treated as a compilation error.

Reporting Tests

To use the TeamCity on-the-fly test reporting, a testing framework needs dedicated support for this feature to work (alternatively, [XML Report Processing](#) can be used).

If TeamCity doesn't support your testing framework natively, it is possible to modify your build script to report test runs to the TeamCity server using service messages. This makes it possible to display test results in real-time, make test information available on the **Tests** tab of the [Build Results](#) page.

Supported test service messages

Test suite messages:

Test suites are used to group tests. TeamCity displays tests grouped by suites on **Tests** tab of the [Build Results](#) page and in other places.

```
##teamcity[testSuiteStarted name='suiteName']  
<individual test messages go here>  
##teamcity[testSuiteFinished name='suiteName']
```

All the individual test messages are to appear between `testSuiteStarted` and `testSuiteFinished` (in that order) with the same name attributes.

Nested test reporting

Prior to TeamCity 9.1, one test could have been reported [from within another test](#).

Since TeamCity 9.1, starting another test finishes the currently started test in the same "flow". To still report tests from within other tests, you will need to specify another [flowId](#) in the nested test service messages.

Test start/stop messages:

```
##teamcity[testStarted name='testName' captureStandardOutput='<true/false>']
<here go all the test service messages with the same name>
##teamcity[testFinished name='testName' duration='<test_duration_in_milliseconds>']
```

Indicates that the test "*testName*" was run. If the `testFailed` message is not present, the test is regarded successful, where

- **duration** (optional numeric attribute) - sets the test duration in milliseconds (should be an integer) to be reported in TeamCity UI. If omitted, the test duration will be calculated from the messages timestamps. If the timestamps are missing, from the actual time the messages were received on the server.
- **captureStandardOutput** (optional boolean attribute) - if true, all the standard output (and standard error) messages received between `testStarted` and `testFinished` messages will be considered test output. The default value is false and assumes usage of `testStdOut` and `testStdErr` service messages to report the test output.

All the other test messages (except for `testIgnored`) with the same name attribute should appear between the `testStarted` and `testFinished` messages (in that order).

Currently, the test-related service messages **cannot be output** with Ant's *echo* task until *flowId* attribute is specified.

It is highly recommended to ensure that the pair of `test suite` + `test name` is unique within the build.

For advanced TeamCity test-related features to work, test names should not deviate from one build to another (a single test must be reported under the same name in every build). Include absolute paths in the reported test names is **strongly discouraged**.

Ignored tests:

```
##teamcity[testIgnored name='testName' message='ignore comment']
```

Indicates that the test "*testName*" is present but was not run (was ignored) by the testing framework.

As an exception, the `testIgnored` message can be reported without the matching `testStarted` and `testFinished` messages.

Test output:

```
##teamcity[testStarted name='className.testName']
##teamcity[testStdOut name='className.testName' out='text']
##teamcity[testStdErr name='className.testName' out='error text']
##teamcity[testFinished name='className.testName' duration='50']
```

The `testStdOut` and `testStdErr` service messages report the test's standard and error output to be displayed in the TeamCity UI. There must be only one `testStdOut` and one `testStdErr` message per test.

An alternative but a less reliable approach is to use the `captureStandardOutput` attribute of the `testStarted` message.

Test result:

```
##teamcity[testStarted name='MyTest.test1']
##teamcity[testFailed name='MyTest.test1' message='failure message' details='message and stack tr
##teamcity[testFinished name='MyTest.test1']

##teamcity[testStarted name='MyTest.test2']
##teamcity[testFailed type='comparisonFailure' name='MyTest.test2' message='failure message' deta
##teamcity[testFinished name='MyTest.test2']
```

Indicates that the "*testname*" test failed. Only one `testFailed` message can appear for a given test name.

- `message` contains the textual representation of the error
- `details` contains detailed information on the test failure, typically a message and an exception stacktrace.
- `actual` and `expected` attributes can only be used together with `type='comparisonFailure'` to report comparison failure. The values will be used when opening the test in the IDE.

Here is a longer example of test reporting with service messages:

```
##teamcity[testSuiteStarted name='suiteName']
##teamcity[testSuiteStarted name='nestedSuiteName']
##teamcity[testStarted name='package_or_namespace.ClassName.TestName']
##teamcity[testFailed name='package_or_namespace.ClassName.TestName' message='The number should b
##teamcity[testFinished name='package_or_namespace.ClassName.TestName']
##teamcity[testSuiteFinished name='nestedSuiteName']
##teamcity[testSuiteFinished name='suiteName']
```

Interpreting test names

A full test name can have a form of: <suite name>:<package/namespace name>.<class name>.<test name>,

where <class name> and <test name> can have no dots in the names.

The [Tests tab of the Build Results page](#) allows grouping by suites, packages/namespaces, classes, and tests. Usually the attribute values are provided as they are reported by your test framework and TeamCity is able to interpret which part of the reported names is the test name, class, package as follows:

TeamCity takes the **suite name** from the corresponding suite message and

- if the reported test name starts with the suite name, the suite name is truncated from the test name before further processing
- the part of the test name *after* the last dot is treated as a **test name**
- the part of the test name *before* the last dot is treated as a **class name**
- the rest of the test name is treated as a **package/namespace name**

Reporting .NET Code Coverage Results

You can configure .NET coverage processing by means of service messages. To learn more, refer to [Manually Configuring Reporting Coverage](#) page.

Publishing Artifacts while the Build is Still in Progress

You can publish the build artifacts while the build is still running, immediately after the artifacts are built.

To do this, you need to output the following line:

```
##teamcity[publishArtifacts '<path>']
```

The <path> has to adhere to the same rules as the [Build Artifact specification](#) of the Build Configuration settings. The files matching the <path> will be uploaded and visible as the artifacts of the running build.

The message should be printed after all the files are ready and no file is locked for reading.

Artifacts are uploaded in the background, which can take time. Please make sure the matching files are not deleted till the end of the build (e.g. you can put them in a directory that is cleaned on the next build start, in a [temp directory](#) or use [Swabra](#) to clean them after the build.)

The process of publishing artifacts process can affect the build because it consumes network traffic and some disk/CPU resources (should be pretty negligible for not large files/directories).

Artifacts that are specified in the build configuration setting will be published as usual.

Reporting Build Progress

You can use special progress messages to mark long-running parts in a build script. These messages will be shown on the projects dashboard for the corresponding build and on the [Build Results page](#).

To log a single progress message, use:

```
##teamcity[progressMessage '<message>']
```

This progress message will be shown until another progress message occurs or until the next target starts (in case of Ant builds).

If you wish to show a progress message for a part of a build only, use:

```
##teamcity[progressStart '<message>']  
...some build activity...  
##teamcity[progressFinish '<message>']
```

The same message should be used for both `progressStart` and `progressFinish`. This allows nesting progress blocks. Also note that in case of Ant builds, progress messages will be replaced if an Ant target starts.

Reporting Build Problems

To fail a build directly from the build script, a build problem has to be reported. Build problems appear on the [Build Results](#) page and also affect the build status text.

To add a build problem to a build, use:

```
##teamcity[buildProblem description='<description>' identity='<identity>']
```

where:

- **description** - (mandatory) a human-readable plain text describing the build problem. By default, the **description** appears in the build status text and in the list of build's problems. The text is limited to 4000 symbols, and will be truncated if the limit is exceeded.
- **identity** - (optional) a unique problem id. Different problems must have different identity, same problems - same identity, which should not change throughout builds if the same problem occurs, e.g. the same compilation error. It should be a valid Java id up to 60 characters. If omitted, the **identity** is calculated based on the **description** text.

Reporting Build Status

TeamCity allows changing the **build status text** from the build script. Unlike [progress messages](#), this change persists even after a build has finished.

You can also change the build status of a failing build to **success**.

Prior to TeamCity 7.1, this service message could have been used for changing build status to **failed**. Since TeamCity 7.1, the [buildProblem](#) service message should be used for that.

To set the status and/or change the text of the build status (for example, note the number of failed tests if the test framework is not supported by TeamCity), use the `buildStatus` message with the following format:

```
##teamcity[buildStatus status='<status value>' text='{build.status.text} and some aftertext']
```

where:

- **status** attribute is optional and may take the value **SUCCESS**.
- **text** attribute sets the new build status text. Optionally, the text can use `{build.status.text}` substitution pattern which represents the status, calculated by TeamCity automatically using passed test count, compilation messages and so on.

The status set will be presented while the build is running and will also affect the final build results.

Reporting Build Number

To set a custom build number directly, specify a `buildNumber` message using the following format:

```
##teamcity[buildNumber '<new build number>']
```

In the `<new build number>` value, you can use the `{build.number}` substitution to use the current build number automatically generated by TeamCity. For example:

```
##teamcity[buildNumber '1.2.3_{build.number}-ent']
```

Adding or Changing a Build Parameter

By using a dedicated service message in your build script, you can dynamically update build parameters of the build right from a build step (the parameters need to be defined in the [Parameters section](#) of the build configuration).

The changed build parameters will be available in the build steps following the modifying one. They will also be available as build parameters and can be used in the dependent builds via `%dep.*% parameter references`.

```
##teamcity[setParameter name='ddd' value='fff']
```

When specifying a build parameter's name, mind the prefix:

- **system** for system properties.
- **env** for environment variables.
- no prefix for configuration parameter.

[Read more about build parameters and their prefixes.](#)

Reporting Build Statistics

In TeamCity, it is possible to configure a build script to report statistical data and then display the charts based on the data. Please refer to the [Customizing Statistics Charts#customCharts](#) page for a guide to displaying the charts on the web UI. This section describes how to report the statistical data from the build script via service messages. You can publish the build statics values in two ways:

- Using a service message in a build script directly
- [Providing data using the teamcity-info.xml file](#)

To report build statistics using service messages: Specify a `'buildStatisticValue'` service message with the following format for each statistics value you want to report:

```
##teamcity[buildStatisticValue key='<valueTypeKey>' value='<value>']
```

where

- The key should not be equal to any of [predefined keys](#).
- The value should be a positive/negative integer of up to 13 digits; float values with up to 6 decimal places are also supported.

Disabling Service Messages Processing

If you need for some reason to disable searching for service messages in the output, you can disable the service messages search with the messages:

```
##teamcity[enableServiceMessages]  
##teamcity[disableServiceMessages]
```


Any messages that appear between these two are not parsed as service messages and are effectively ignored.

For server-side processing of service messages, enable/disable service messages also supports the flowId attribute and will ignore only the messages with the same flowId.

Importing XML Reports

In addition to the [UI Build Feature](#), XML reporting can be configured from within the build script with the help of the `importData` service message.

Also, the message supports importing of the previously collected code coverage and code inspection/duplicates reports.

The service message format is:

```
##teamcity[importData type='typeID' path='<path to the xml file>']
```

To be processed, report XML files (or a directory) must be located in the checkout directory, and the path must be relative to [this directory](#).

where typeID can be one of the following (see also [XML Report Processing](#)):

| typeID | Description |
|------------------------------------|---|
| Testing frameworks | |
| junit | JUnit Ant task XML reports |
| surefire | Maven Surefire XML reports |
| nunit | NUnit-Console XML reports |
| mstest | MSTest XML reports |
| vstest | VSTest XML reports |
| gtest | Google Test XML reports |
| Code inspection | |
| checkstyle | Checkstyle inspections XML reports |
| findBugs ²⁾ | FindBugs inspections XML reports |
| jslint | JSLint XML reports |
| ReSharperInspectCode ¹⁾ | ReSharper inspectCode.exe XML reports |
| FxCop ¹⁾ | FxCop inspection XML reports |
| pmd | PMD inspections XML reports |
| Code duplication | |
| pmdCpd | PMD Copy/Paste Detector (CPD) XML reports |
| DotNetDupFinder ¹⁾ | ReSharper dupfinder.exe XML reports |
| Code coverage | |
| dotNetCoverage ^{1) 3)} | XML reports generated by dotcover, partcover, ncover or ncover3 |

Notes:

¹⁾ only supports specific file in the path attribute

²⁾ also requires the `findBugsHome` attribute specified pointing to the home directory of the installed FindBugs tool.

³⁾ also requires the `tool=<tool name>` service message attribute, where the `<tool name>` is either `dotcover`, `partcover`, `ncover` or `ncover3`.

If not specially noted, the report types support Ant-like wildcards in the `path` attribute.

the `verbose='true'` attribute will enable detailed logging into the build log.

the `parseOutOfDate='true'` attribute will process all the files matching the path. By default, only those updated during the build (determined by last modification timestamp) are processed. If any not matching reports are found, the "report skipped as out-of-date" message appears in the build log.

the `whenNoDataPublished=<action>` (where `<action>` is one of the following: `info` (default), `nothing`, `warning`, `error`) will change output level if no reports matching the path specified were found.

(deprecated, use [Build Failure Conditions](#) instead)

`findBugs`, `pmd` or `checkstyle` `importData` messages also take optional `errorLimit` and `warningLimit` attributes specifying errors and warnings limits, exceeding which will cause the build failure.

After the `importData` message is received, TeamCity agent starts to monitor the specified paths on the disk and imports matching report files in the background as soon as the files appear on disk.

The parsing only occurs within the build step in which the messages were received. On the step finish, the agent ensures all the present reports are processed before beginning the next step. This behavior is different from that of [XML Report Processing](#) build feature, which completes files parsing only at the end of the build.

Please ensure the report files are available after the generation process ends (the files are not deleted, nor overwritten by the build script)

To initiate monitoring of several directories or parse several types of the report, send the corresponding service messages one after another.

Only several reports of different types can be included in a build. Processing reports of several inspections or duplicates tools in a single build is not supported. See the [related feature request](#).

Libraries reporting results via TeamCity Service Messages

Several platform-specific libraries from JetBrains and external sources are able to report the results via TeamCity Service messages.

- [Service messages .NET library](#) - .NET library for generating (and parsing) TeamCity service messages from .NET applications. See a [related blog post](#).
- [Jasmine 2.0 TeamCity reporter](#) - support for emitting TeamCity service messages from Jasmine 2.0 reporter
- [Perl TAP Formatter](#) - formatter for Perl to transform TAP messages to TeamCity service messages
- [PHPUnit 5.0](#) - supports TeamCity service messages for tests. For earlier PHPUnit versions, the following external libraries can be used: [PHPUnit Listener 1](#), [PHPUnit Listener 2](#) - listeners which can be plugged via PHPUnit's `suite.xml` to produce TeamCity service messages for tests.
- [Python Unit Test Reporting to TeamCity](#) - the package that automatically reports unit tests to the TeamCity server via service messages (when run under TeamCity and provided the testing code is adapted to use it).
- [Mocha](#) - on-the-fly reporting via service messages for Mocha JavaScript testing framework. See the related [post](#) with instructions.
- [Karma](#) - support in the JavaScript testing tool to report tests progress into TeamCity using TeamCity service messages

teamcity-info.xml

As an obsolete approach, it is also possible to have the build script collect information, generate an XML file called `teamcity-info.xml` in the root build directory. When the build finishes, this file will automatically be uploaded as a build artifact and processed by the TeamCity server.

Please note that this approach **can be discontinued** in the future TeamCity versions, so service messages approach is recommended instead. In case service messages does not work for you, please let us know the details and describe the case via [email](#).

Modifying the Build Status

TeamCity has the ability to change the build status directly from the build script. You can set the status (build failure or success) and change the text of the build status (for example, note the number of failed tests if the test framework is not supported by TeamCity).

[XML schema for teamcity-info.xml](#)

It is possible to set the following information for the build:

Build number — Sets the new number for the finished build. You can reference the TeamCity-provided build number using {build.number}.

Build status — Change the build status. Supported values are "FAILURE" and "SUCCESS".

Status text — Modify the text of build status. You can replace the TeamCity-provided status text or add a custom part before or after the standard text. Supported action values are "append", "prepend" and "replace".

Example teamcity-info.xml file:

```
<build number="1.0.{build.number}">
  <statusInfo status="FAILURE"> <!-- or SUCCESS -->
    <text action="append"> fitness: 45</text>
    <text action="append"> coverage: 54%</text>
  </statusInfo>
</build>
```

It is up to you to figure out how to retrieve test results that are not supported by TeamCity and accurately add them to the teamcity-info.xml file.

Reporting Custom Statistics

It is possible to provide [custom charts](#) in TeamCity. Your build can provide data for such graphs using teamcity-info.xml file.

Providing data using the teamcity-info.xml file

This file should be created by the build in the root directory of the build. You can publish multiple statistics (see the details on the data format below) and create separate charts for each set of values.

The teamcity-info.xml file is to contain the code in the following format (you can combine various data in the teamcity-info.xml file):

```
<build>
  <statisticValue key="chart1Key" value="342"/>
  <statisticValue key="chart2Key" value="53"/>
</build>
```

The key should not be equal to any of [predefined keys](#).

The value should be a positive/negative integer of up to 13 digits. **Since TeamCity 9.0**, float values with up to 6 decimal places are also supported.

The key here relates to the key of the **valueType** tag used when describing the chart.

Describing custom charts

See [Customizing Statistics Charts](#) page for detailed description.

30 Comments

**Alexander V. Ilyin**

Message FlowId could be used in any service message? I want 'spam' tests in 8 flows to reduce testing time and TC will consume data as expected?

**Alexander V. Ilyin**

Seem it works as I expect but sometimes service messages could be split by some other text but I fixes it by redirecting some output to file in Ant's exec task.

**Matt Davis**

What is the difference between the following in terms:##teamcity[]...some build activity...##teamcity[] and##teamcity[]...some build activity...##teamcity[]

**Victory Petrenko**

What is the difference between the following in terms:##teamcity[]...some build activity...##teamcity[] and##teamcity[]...some build activity...##teamcity[]

Matt, progress messages are not only added to the Build log, but to the build status text, which means that if you open a progress block it will be present in the build status text until you close it.

**Anton Rudeshko**

Is there a name for parameter in single attribute messages?

Generally, is there any way to convert single attribute message into multiple attribute message?

**Victory Petrenko**

Anton, no there is no hidden name for a default attribute. What is your case and what message are you using?

**Anton Rudeshko**

I have no particular case, just want to know if there is any single format that fits both types of messages. Thank you, Vika

**Pavel Miroshnichenko**

Could you add one more message - "testSuiteFailed"?testSuiteStarted, testSuiteFinished and testSuiteFailed

If one of tests has been failed, I need mark testSuite as failed and stop it (and subtests). After this action, I'm going to another testSuite.

Thanks!

**Victory Petrenko**

Pavel, looks like there is no sense in adding such a service message as TeamCity doesn't support any suite failures and won't interpret it or show somehow.

**Marc Abramowitz**

It would be cool if there was a service message that let you change the name of a tab.

In my use case, we create a tab called "Warnings" and various tools write warnings into an HTML file which that tab shows. It would be cool if I could send a service message that changes the tab name from "Warnings" to "Warnings (42)" or whatever. That would let users know whether they should bother to click the tab.

I guess this could also be solved by adding support for warnings directly to TeamCity. It seems that users currently have to hack their own solution; it would be great if TeamCity could standardize this.

**Marc Abramowitz**

Does TeamCity do anything special when I use:

```
##teamcity[message text='This is a warning' status='WARNING']
```

?

I don't see any way to see all of the build warnings. It would be great if there were a tab that showed all the warnings.

**Julia Alexandrova**

Marc, this service message will mark a build log message as a warning. TeamCity will not process them separately. Have you tried [this approach](#)?

**Marc Abramowitz**

Yes, I used `buildProblem` to log **fatal** problems that fail the build.

But we also have certain checks we do for style things or for people using deprecating things and for those, I'd like to emit a **non-fatal** warning. We started by using:

```
##teamcity[message text='This is a warning' status='WARNING']
```

but the problem with that is that those are just in the build log and were too easy for developers to ignore.

So now we have a function that does that but also appends the warning to a file and then we've configured TeamCity to show a tab for that file and rigged up counting of warnings so we can graph them over time. That works pretty well, but I am thinking that it would be great if TeamCity could just add a feature for this.

I am not the only one who has missed this feature:

- <http://blogs.lessthandot.com/index.php/enterprisedev/application-lifecycle-management/displaying-net-build-warnings-in/>
- <http://www.nichesoftware.co.nz/2008/12/30/tracking-warnings-with-teamcity.html>
- <http://homenko.net/post/2013/11/22/Enhanced-MSBuild-Warning-report-in-TeamCity>
- <http://stackoverflow.com/questions/19587082/teamcity-reporting-of-visual-studio-warnings>

**Julia Alexandrova**

Marc, thank you very much for all your comments. It sounds like it could be a useful feature indeed. Could you please add a feature request to our [tracker](#)?

**Marc Abramowitz**

Yep. <https://youtrack.jetbrains.com/issue/TW-42729>

Thanks!

**Marc Abramowitz**

It would be useful to see screenshots of how TeamCity renders the various service messages. For example, what is the difference between `compilationStarted` vs. `blockOpened`?

**Julia Alexandrova**

Marc, I am not sure screenshots will be explanatory in this case. Both messages you mention are used to open a block of service messages in the build log, both should be closed with a corresponding closing message. The only difference is in processing the errors inside the blocks - any error reported between `compilationStarted` and `compilationFinished` messages will be treated as a compilation error, i.e. a critical build problem.

**Marc Abramowitz**

It might be a good idea to add a section with links to language-specific libraries for emitting service messages – e.g.:

- <https://pypi.python.org/pypi/teamcity-messages>
- <https://github.com/JetBrains/TeamCity.ServiceMessages>
- <https://www.npmjs.com/package/teamcity-service-messages>
- <http://search.cpan.org/~thaljef/TAP-Formatter-TeamCity-0.04/lib/TAP/Formatter/TeamCity.pm>

Here's a snippet of the bash that we're using:

```
emit_teamcity_block_opened "Adding number of warnings to build status"
local num_warnings=$(egrep -ic '^(warn|warning):' pipeline_warnings.html)
teamcity_set_build_status "{build.status.text}; Warnings: $num_warnings"
teamcity_build_statistic_value "python_pipeline:num_warning" $num_warnings
emit_teamcity_block_closed "Adding number of warnings to build status"
```

It would be great if TeamCity tracked the warning automatically, so I didn't have to do this.

**Julia Alexandrova**

@Marc Abramowitz, I added [a section](#) to his page. Thanks for the tip!

**Marc Abramowitz**

Cool, thanks!

**Marc Abramowitz**

When using buildStatus, is there a way to get the Build Overview page to show the status with just the text without the parent blocks?

Let me illustrate the problem with a screenshot:

(moderator: screenshots on S3 are insecure, deleted)

Note how the **Running Step** is extremely long, because it is showing the whole block hierarchy which can be very long if there are a lot of nested blocks (or just a few nested blocks that have long descriptions).

I'd like to not show all of the block hierarchy and just show the simple step name, which in this case is "get status of all processes", which is actually partially cut off above, because the step name is so long.

Interestingly, in at least one other place in TeamCity, it shows the shorter and more readable step name:

(moderator: screenshots on S3 are insecure, deleted)

**Julia Alexandrova**

Unfortunately, there is no standard way of cutting off the beginning blocks. Please post an issue to [our tracker](#).

**Marc Abramowitz**

Done.

<https://youtrack.jetbrains.com/issue/TW-42732>

Thanks!

**Ewin Hong**

Hello,

Here is my question.

So I am using Team City and I have my Automated UI Tests running on a command line build step to execute the ArtOfTest runner from Telerik (<http://docs.telerik.com/teststudio/features/test-runners/artoftest-runner>).

Below is a couple lines from the build log.

The command line does executes all of the provided tests, but it does not identify the individual tests.

So is adding Service messages the way to identify the test?

The line that specifically says Executing test: '607...' is the clear indication of when the Test starts.

Then the line following this like Execute Test... is where a subtest for login and navigation.

So are service messages applicable for this type of use in telling me when a test starts, end, and fails?

```
[16:50:36]Step 2/2: Run QA Tests Art of Test Runner Command Line admin (Command Line) (1h:26m:56s)
[16:50:37][Step 2/2] Starting: "C:\Program Files (x86)\Telerik\Test Studio\Bin\artoftest.runner.exe" list="C:\Users\lewin.hong\
\Desktop\buildAgent\work\4356906eeffb0cad\MyVector.AutomatedTest\testlists\myvector_dev.aiilist"
[16:50:37][Step 2/2] in directory: C:\Users\lewin.hong\Desktop\buildAgent\work\4356906eeffb0cad
[16:50:46][Step 2/2] Executing test: '607 U V Update My Profile Page'
[16:50:48][Step 2/2] Execute test 'Load MyVector'
[16:50:48][Step 2/2] Navigate to : '/'
[16:51:25][Step 2/2] IF (Verify ExistsNot 'OverridelinkLink')
[16:51:26][Step 2/2] Wait '5000' msec for url:'MyVector'
[16:51:26][Step 2/2] Click 'UpdateMyH4Tag'
```

**Julia Alexandrova**

@Ewin, yes, service messages can be used to to identify an individual test, and to report to TeamCity the information on a test start, finish, and the test status. See details [here](#).

**Ewin Hong**

@Julia Alexandrova,

To be honest, I read this documentation over and over several times. But I guess I am missing a piece of information to be effective in what I am trying to achieve.

```
##teamcity[testStarted name='testName' captureStandardOutput='<true/false>']
```

In the snippet here, I would like to pass in "607 U V Update My Profile Page" coming from this line into the name parameter.

```
[16:50:46][Step 2/2] Executing test: '607 U V Update My Profile Page'
```

Is this possible? Or is there another way to achieve my intent?

**Julia Alexandrova**

@Ewin, the [forum](#) is actually better suited to ask for assistance. The documentation does not seem to cover your needs, so could you please post all the details into the [forum](#)? Thank you.

**Sam Thursfield**

Adding or Changing a Build Parameter

By using a dedicated service message in your build script, you can dynamically update build parameters of the build right from a build step.

The changed build parameters will be available in the build steps following the modifying one. They will also be available as build parameters and can be used in the dependent builds via `%dep.*%parameter references`.

```
##teamcity[setParameter name='ddd' value='fff']
```

When specifying a build parameter's name, mind the prefix:

- **system** for system properties.
 - **env** for environment variables.
 - no prefix for configuration parameter.
- [Read more about build parameters and their prefixes.](#)

It would be really useful if this section mentioned that you need to define the parameter in the 'Parameters' section of the build config. This feature is nice, but I tried to set a parameter and then use it, and I hit the error where builds no longer start saying "No compatible agents." It turned out that I needed to define the property in the 'Parameters' section of the build config, but it took over an hour to work this out.

**Julia Alexandrova**

Thanks, [Sam](#). I added the note to the [section](#).

**Tim Baverstock**

It would be useful if a message could ask for this build to be rescheduled on a different machine: our MacOS devices frequently 'lose' their Android phones, despite running adb kill-server, and occasionally the phones mysteriously power off overnight. Transferring the build to a randomly chosen other agent would be better than a failed build.